Simulation of Real-Time Scheduling with Various Execution Time Models

Maxime Chéramy, Pierre-Emmanuel Hladik, Anne-Marie Déplanche, Sébastien Dubé

Simulation of Real-Time Scheduling with Various Execution Time Models

Maxime Chéramy*, Pierre-Emmanuel Hladik*, Anne-Marie Déplanche† and Sébastien Dubé‡
*CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France
Univ de Toulouse, INSA, LAAS, F-31400 Toulouse, France
†IRCCyN UMR CNRS 6597, (Institut de Recherche en Communications et Cybernétique de Nantes), ECN,
1 rue de la Noe, BP92101, F-44321 Nantes cedex 3, France
‡Hella Engineering France S.A.S.
4 rue du Professeur Vellas, 31300 Toulouse, France

***Abstract*—In this paper, we present SimSo, a simulator that aims at facilitating the design of experimental evaluations for real-time scheduling algorithms. Currently, more than twenty-five algorithms were implemented. Special attention is paid to the execution time model of tasks. We show that the worst-case execution time for experimental simulation can introduce a bias in evaluation and we discuss as a work in progress how cache effects could be taken into consideration in the simulation.**

I. Introduction

Davis and Burns referenced more than thirty real-time scheduling algorithms in 2011 [7] and more than a dozen of new algorithms have emerged since then. Such a large number of scheduling algorithms makes their evaluation and comparison difficult. The evaluation generally comes from theoretical analysis, simulation or an actual implementation, according to criteria that can include utilization bounds, success rates, number of preemptions, number of migrations, and/or algorithm complexity.

This evaluation is difficult by using theoretical analyses such as schedulability tests or resource augmentation. On the other hand, while using a real system would seem to be a better approach, the effective implementation of a scheduler as an operating system component requires a substantial amount of time and the results are specific to the considered system. As a consequence, we think that simulation is a good compromise to efficiently evaluate scheduling algorithms.

Contribution. This paper briefly presents SimSo, a tool that simulates real-time schedulers. SimSo aims at facilitating the design of experimental evaluations: its interface makes possible to implement the scheduling algorithms in a realistic way; some tools enable to design experiments by defining and collecting the measures; and various tasksets generators are provided.

Our long-term objective is to compare the various schedulers while taking into account the effect of the hardware architecture (e.g. caches, dynamic frequency scaling, or system overheads) on their performance. These hardware components have an effect on the computation time of the jobs and, as a consequence, a way to model these effects through the simulation is required. SimSo propose a flexible way to define various models of execution time that we discuss. We show the effect of worst-case execution time model on experimental results and we discuss as a work in progress how cache effects could be taken into consideration in the simulation.

Paper organization. The remainder of this paper is organized as follows: in Section II, related work is recalled. Section III gives a short insight of SimSo. Section IV discusses the bias introduced by the worst-case execution time assumption and the possibility to introduce directly the cache effects in the simulation. Finally, Section V provides some concluding remarks and envisages future work.

II. Related Work

Davis and Burns exposed in [7] four performance metrics to compare the effectiveness of various scheduling algorithms: (i) utilization bounds: the bound on utilization factor to guarantee the schedulability of a system; (ii) approximation ratio: comparison of the performance of an algorithm with an optimal one; (iii) resource augmentation: the gap on processor speed to fulfill the schedulability; and (iv) empirical measures: evaluation of the performance of scheduling algorithms on randomly generated tasksets. Our work addresses metric iv.

Empirical evaluations of scheduling algorithms focus on the overheads involved in scheduling decisions. The main studied causes of overheads are context switches, preemptions, migrations and computational complexity. Two approaches are typically considered to evaluate them. The first one is based on performance measured on a real platform with a dedicated operating system, e.g. the experiments done with LITMUS^{RT} [4], an extension of the Linux Kernel developed at the University of North Carolina, or the experimental work of Lelli et al. [13] on a dedicated implementation of Linux with RM and EDF-based multiprocessor schedulers. This method could also be conducted on a cycle-accurate simulated architecture with a real operating system as in [19]. The second approach is to use tools dedicated to the simulation of real-time systems. Most of these tools are designed to validate, test and analyze systems. MAST [9] proposes a set of tools to model and analyze distributed real-time systems and it also includes a simulator, JSimMAST. Cheddar [17] proposes a GUI comprising a simulator, many feasibility tests and it is also used to simulate AADL models. STORM [18] and YARTISS [5] offer a simulator to conduct evaluation on scheduling algorithms with the possibility to easily join new scheduling policies.

III. SIMSO

To facilitate the experimentation of scheduling algorithms, we thus propose a dedicated and open source tool: SimSo[1], a real-time scheduling simulator designed to be easy to use as well as extend [6]. SimSo simulates the execution of a taskset on one or multiple processors, where a taskset consists of periodic tasks defined by their period, deadline, worst-case execution time and eventually additional parameters.

A. Architecture

The core of SimSo relies on SimPy [16], a process-based discrete-event simulation framework. The design of SimSo is influenced by real systems: there are processors, tasks, jobs, timers, etc. The instances of *Processors* are the central part of the simulation because they simulate the functional behavior of a processor and of the operating system. Each processor can execute a job or be interrupted to execute a method of the scheduler. The design of SimSo and the use of discrete-event simulation allow us to set various time overheads to simulate the context-switches, scheduler calls, locks, etc. Such overheads are applied on the processor they are supposed to occur.

B. Writing a Scheduler

One of the advantages of using a simulator is to simplify the experimentation. Writing a scheduler should therefore be as easy as possible and rely on useful methods. A scheduler for SimSo is a Python class that inherits from the *Scheduler* class and is loaded dynamically into the simulator. In practice, most of the schedulers are implemented with less than 200 lines of code. The language is different to the one that would be used on a real implementation, however, this does not change the underlying algorithms and logic.

The scheduler interface in SimSo is partly influenced by real operating systems, but kept as simple as possible. Its similarity with a realistic system allows us to raise practical issues regarding the implementation that could have been hard or even impossible to integrate into theoretical studies. For instance, we need to decide which processor should run the scheduler and this may have an impact on the performance or even the schedulability. Another example is the finite precision of the timers: this may introduce a tiny difference compared to the theoretical schedule and cause a major issue.

We have already implemented more than twenty-five schedulers including: uniprocessor schedulers with RM, DM, FP, EDF and M-LLF; *partitioned* approach such as P-EDF and P-RM; *global* algorithms: G-RM, G-EDF, G-FL, EDF-US, PriD, EDZL, M-LLF and U-EDF; *PFair* schedulers with PD^2 and its work-conserving variant ER-PD^2; *BFair* and *DP-Fair* variants with LLREF, LRE-TL, DP-WRAP, BF and NVNLF; *semi-partitioned* approaches with EDHS, EKG and RUN; *DVFS* schedulers with Static-EDF and CC-EDF.

C. Execution Time Model

When simulation is used to study the schedulability of a system, it is usual that the tasks meet their worst-case execution time at each job. However, as explained in section IV-A, this is actually very pessimistic and it is not a realistic assumption. A a consequence, the Liu and Layland model should be adapted: for instance, the multiframe model is a way to tackle this problem [15].

Also, many scheduling evaluations only focus on the number of preemptions and migrations because they are the source of overheads. A preemption induces a system overhead due to the context-switching, but it may also increase the computation time of a job by causing extra cache misses. To increase realism, it would be interesting to integrate Cache-Related Preemption Delays within the computation time of the jobs. This is even mandatory for the evaluation of cache-aware schedulers.

As a consequence of the two previous remarks, it is desirable to have the possibility to simulate a system with customized durations of jobs, depending on the purpose of the simulation. Several *Execution Time Models* (ETM) are already available in SimSo and others could be added. The simplest model consists of using the WCET of the tasks for their execution time. A second one uses a random duration for each job to meet a given average execution time (ACET). The ACET model uses a normal distribution defined by its mean, its standard deviation and is bounded by the WCET. Another model detects the jobs that have undergone a preemption or a migration, and extends their WCET[2] using fixed time penalties. Finally, a more complex model tries to simulate the state of the caches. In this latter model, the execution time of the jobs depends on the events that happen during their activation period. This ETM is also interesting because it simulates the impact of shared caches and, as a consequence, it is impossible to know in advance when a job will end since it depends on external events (see Section IV-B).

The first three models can also deal with Dynamic Voltage and Frequency Scaling (DVFS). The current DVFS model simply considers that a job consumes its computation time proportionally to the processor speed. This is obviously a simplified assumption, but it is possible to implement more realistic ETM models to deal with DVFS.

D. Generation of Tasksets and Collecting Simulation Results

The tasksets can be manually specified by the user or be generated by SimSo. Indeed, in order to ease the work of the experimenter, SimSo provides several methods to generate the tasksets. The method presented by Kato et al generates a taskset defined by its total utilization but with a variable number of tasks [11] while the algorithms UUniFast-Discard and RandFixedSum generate a taskset with a given number of tasks and total utilization [8]. These algorithms are combined with various period generators: uniform, log-uniform or among a set of periods. It is also possible to use other methods that are not provided by SimSo.

During the simulation of a system with SimSo, every significant events are traced. Whereas this approach is actually heavier than just counting events such as the preemptions and migrations during the simulation, it provides more flexibility.

[1] SimSo: http://homepages.laas.fr/mcheramy/simso/

[2] In this case, the WCET is defined as the worst-case execution time without any interruption.

A set of methods are available to ease the retrieval of usual metrics such as success rate, number of preemptions and migrations, number of scheduler calls, laxity, etc. And it is also possible to post-treat the trace in order to collect some specific data.

SimSo provides a graphical user interface that helps to configure a system and run it. That GUI is capable of displaying common measures such as preemptions, migrations, or execution times. It is also possible to display a gantt chart, which is very useful during the development of a scheduler. However, this GUI only shows the results for a single simulation but the simulations can be fully-automated using Python scripts.

IV. Discussion about Execution Time Models

A recurring criticism against the use of simulation is the lack of realism. This is probably due to over-simplified task and architecture models. But we also believe that simulation is a convenient and flexible way to conduct large evaluations. As a consequence, we are willing to make the simulation more realistic. In this section, we are discussing the use of the WCET in the simulation and how we could integrate the cache effects.

A. Worst-Case Execution Time

In the literature, the empirical measures are conducted such that all tasks meet their worst-case execution time at each job. However, the use of the WCET is in fact very pessimistic: the worst-case is an upper-bound and the jobs of a task never consume all this time; also, the active jobs never meet their WCET all at the same time.

This assumption could induce an erroneous evaluation of the scheduling algorithms performances. For instance, some algorithms are naturally robust to a punctual overload and can adapt their response to this demand. Moreover, this assumption could give an advantage to the scheduling algorithms that use the WCET as a parameter and highly depend on it. It seems reasonable to study the performance of scheduling policies not only in the worst-case situation, but also with variable execution times.

As explained in section III-C, the execution time model in SimSo can be easily implemented in various ways. To show the impact of variations in the execution time, a Gaussian random behavior bounded by the worst-case response time is implemented. The expectation and the standard deviation of the task durations are fixed relatively to the worst-case execution time. For each job, a computation time is randomly generated to respect an average-case response time (ACET) and bounded by the worst-case execution time.

The figure 1 shows results with an ACET equal to 75% of the WCET for one hundred tasksets scheduled between 0-1000ms with EDF, RUN and U-EDF schedulers. Each taskset contains 50 synchronous periodic tasks with implicit deadlines. These tasks are scheduled on 4 processors for a total utilization between 75% and 97.5%. The periods of the tasks are chosen using a log-uniform distribution on the range 2-100ms and rounded to the closest integer. Their utilization factor is generated using the RandFixedSum algorithm. The execution time of each job is equal to the product of the period and the utilization factor.

The experiments named EDF, RUN and U-EDF are obtained using the WCET for the computation times of the jobs, and experiments EDF-A, RUN-A and U-EDF-A use ACET on the same tasksets. The effective utilization with ACET is given by the top axis. We show that EDF is not sensitive to the WCET parameter: for the same effective utilization, WCET and ACET experiments give the same number of preemptions. If we only consider WCET, RUN does approximatively 30% less preemptions than U-EDF for a utilization of 95%. However, if we consider ACET for the same system, they are quite similar. It exhibits the fact that the scheduling decisions made by the RUN algorithm are more affected by the WCET parameter than U-EDF. Such an observation suggests us some possible improvements to the RUN algorithm that will be easily assessed with SimSo.

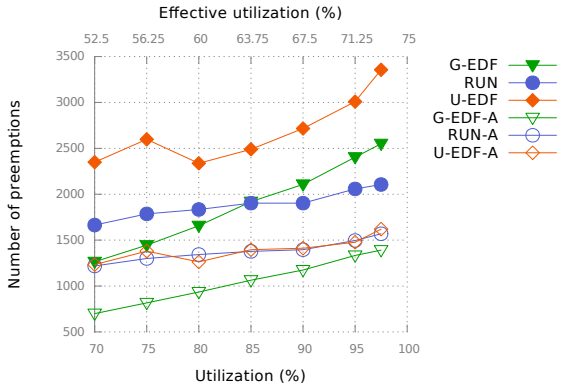

Fig. 1. Number of preemptions (including migrations) for G-EDF, RUN and U-EDF using the WCET and the ACET.

Remark that during the design of an industrial application, WCET is rarely used to evaluate the schedulability of an application. Konig et al. explain in [12] that engine control applications are not schedulable using classic methods with WCET, but the systems are still effective in practice. Although the WCET is useful to have a theoretical evaluation of the schedulability for hard real-time systems, these practical cases invite criticism about the use of WCET as parameter in scheduling algorithms and in the evaluation of their performance.

For all these reasons, we think that the experimental evaluations that usually rely on the WCET should be completed by other experiments that use more realistic computation times.

B. Influence of Caches

As mentioned in Section III-C, many experimental studies focus on the number of preemptions and migrations because they are the sources of runtime overheads. Mogul and Borg [14] showed that the cache related preemption delay is actually greater that the direct overheads caused by the operating system.

Moreover, on multiprocessor architectures with shared caches, the parallel execution of jobs may have an influence on the general performance of the system. Several studies have shown that avoiding co-scheduling tasks that heavily use a shared cache can reduce the overall execution time, e.g. the concept of megatasks [1]. Finally, other works focus on cache

space isolation techniques to avoid cache contention on shared caches [10], [3].

The impact of a preemption on the caches depends on the programs involved, when it occurs or what happens between the interruption of a job and its resumption. Whereas it is possible in SimSo to add a fixed overhead to simulate a context switch, the present task model does not bring enough information to estimate the real impact of a preemption. The same conclusion applies regarding the influence of shared caches.

Therefore, we want to enrich the Liu and Layland task model with additional informations that would characterize the memory behavior of the tasks. As a matter of fact, other scientific communities have also been studying the influence of caches on the execution of programs. In particular, many statistical cache models aim at evaluating the number of cache misses [2]. Those are often based on the following metrics:

- Stack Distance Profile: A distribution of distances that indicates the probabilities that a cache access is done at a given position in an LRU cache;
- MIX or API: The number of memory reference per instruction;
- CPI: The average number of cycles needed to execute an instruction.

Using this information, the number of instructions and the time penalty associated to a cache miss or hit, it is possible to estimate the computation time of a job. It is then possible to simulate the impact of preemptions, migrations, and also the impact of cache sharing between several jobs.

Based on such statistical models, SimSo is already capable to simulate the impact of (data) caches using data collected from real programs. Such data can be collected from real programs and, in our case, we use the Gem5 simulator in order to do so for various benchmarks (MiBench, Mälardalen, and a few custom programs). However, we are still evaluating the accuracy of the cache models and their use in our context. This is a mandatory step before conducting any reliable experiments. Once this verification is done, that would allow us to study cache-aware schedulers.

V. Conclusion

Our objective is to ease the comparison of the numerous scheduling policies. We have decided to use simulation because it can simplify the implementation of a scheduler while offering more flexibility than a real system would allow. Our main contribution is SimSo, an effective tool to evaluate real-time scheduling algorithms. A particular care has been taken to keep a realistic scheduling interface so that practical decisions are not eluded. At the present time, more than twenty-five schedulers and various tasksets generators are proposed to assist the experimenter. Experiments can be automated with Python scripts to collect and analyze data. Future work includes a large evaluation of the scheduling algorithms and introducing more complex task behaviors such as shared resources and precedence relations.

In the last section, we have discussed the usage of worst-case execution time in experiments. This assumption may not be very realistic but more importantly, schedulers that heavily rely on the WCET could benefit from simulation using WCET. In that sense, we have suggested to conduct experimentations using average-case execution times instead.

We have also shown how the execution time model of SimSo can be extended to take into account cache effects. We do not try to simulate the exact behavior of the cache memory, but modeling their statistical performance and impact on the scheduling of tasks. This work is in progress and we are currently evaluating the accuracy of the results.

Acknowledgment

The work presented in this paper was conducted under the research project RESPECTED (http://anr-respected.laas.fr/) which is supported by the French National Agency for Research (ANR), program ARPEGE.

References

[1] J. Anderson, J. Calandrino, and U. Devi, "Real-time scheduling on multicore platforms," in *Proc. of RTAS*, 2006.

[2] V. Babka, P. Libič, T. Martinec, and P. Tůma, "On the accuracy of cache sharing models," in *Proc. of ICPE*, 2012.

[3] B. Berna and I. Puaut, "PDPA: period driven task and cache partitioning algorithm for multi-core systems," in *Proc. of RTNS*, 2012.

[4] J. Calandrino, H. Leontyev, A. Block, U. C. Devi, and J. Anderson, "LITMUS^{RT}: A testbed for empirically comparing real-time multiprocessor schedulers," in *Proc. of RTSS*, 2006.

[5] Y. Chandarli, F. Fauberteau, D. Masson, S. Midonnet, and M. Qamhieh, "YARTISS: A Tool to Visualize, Test, Compare and Evaluate Real-Time Scheduling Algorithms," in *Proc. of WATERS*, 2012.

[6] M. Chéramy, A.-M. Déplanche, and P.-E. Hladik, "Simulation of real-time multiprocessor scheduling with overheads," in *Proc. of SIMULTECH*, 2013.

[7] R. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Comput. Surv.*, vol. 43, no. 4, 2011.

[8] P. Emberson, R. Stafford, and R. Davis, "Techniques for the synthesis of multiprocessor tasksets," in *Proc. of WATERS*, 2010.

[9] M. Gonzalez Harbour, J. Gutierrez Garcia, J. Palencia Gutierrez, and J. Drake Moyano, "MAST: Modeling and analysis suite for real time applications," in *Proc. of ECRTS '01*, 2001.

[10] N. Guan, M. Stigge, W. Yi, and G. Yu, "Cache-aware scheduling and analysis for multicores," in *Proc. of EMSOFT*, 2009.

[11] S. Kato and N. Yamasaki, "Portioned EDF-based scheduling on multiprocessors," in *Proc. of EMSOFT*, 2008.

[12] F. Konig, D. Boers, F. Slomka, U. Margull, M. Niemetz, and G. Wirrer, "Application specific performance indicators for quantitative evaluation of the timing behavior for embedded real-time systems," in *Proc. of DATE*, 2009.

[13] J. Lelli, D. Faggioli, T. Cucinotta, and G. Lipari, "An experimental comparison of different real-time schedulers on multicore systems," *Journal of Systems and Software*, vol. 85, no. 10, 2012.

[14] J. C. Mogul and A. Borg, "The effect of context switches on cache performance," *SIGPLAN Not.*, vol. 26, no. 4, 1991.

[15] A. K. Mok and D. Chen, "A multiframe model for real-time tasks," in *Proc. of RTSS*, 1996.

[16] SimPy, Online: http://simpy.readthedocs.org/, 2014.

[17] F. Singhoff, J. Legrand, L. Nana, and L. Marcé, "Cheddar: A flexible real time scheduling framework," *Ada Lett.*, vol. XXIV, no. 4, 2004.

[18] R. Urunuela, A.-M. Déplanche, and Y. Trinquet, "STORM a simulation tool for real-time multiprocessor scheduling evaluation," in *Proc. of ETFA*, 2010.

[19] D. Zhu, D. Mosse, and R. Melhem, "Multiple-resource periodic scheduling problem: how much fairness is necessary?" in *Proc. of RTSS*, 2003.